

A Decentralized Quasi-Newton Method for Dual Formulations of Consensus Optimization

Mark Eisen, Aryan Mokhtari, and Alejandro Ribeiro

Abstract—This paper considers consensus optimization problems where each node of a network has access to a different summand of an aggregate cost function. Nodes try to minimize the aggregate cost function, while they exchange information only with their neighbors. We modify the dual decomposition method to incorporate a curvature correction inspired by the Broyden-Fletcher-Goldfarb-Shanno (BFGS) quasi-Newton method. The resulting dual D-BFGS method is a fully decentralized algorithm in which nodes approximate curvature information of themselves and their neighbors through the satisfaction of a secant condition. Dual D-BFGS is of interest in consensus optimization problems that are not well conditioned, making first order decentralized methods ineffective, and in which second order information is not readily available, making decentralized second order methods infeasible. Asynchronous implementation is discussed and convergence of D-BFGS is established formally for both synchronous and asynchronous implementations. Performance advantages relative to alternative decentralized algorithms are shown numerically.

Index Terms—Multi-agent network, consensus optimization, quasi-Newton methods, dual methods

I. INTRODUCTION

We study the problem of decentralized consensus optimization where nodes of a network maximize a global objective function, while each of them has access to a different summand of the global objective function. To be more precise, consider a variable $\tilde{\mathbf{x}} \in \mathbb{R}^p$ and a local strongly concave function $f_i : \mathbb{R}^p \rightarrow \mathbb{R}$ associated with node i . The goal of nodes is to solve the optimization problem

$$\tilde{\mathbf{x}}^* := \operatorname{argmax}_{\tilde{\mathbf{x}} \in \mathbb{R}^p} f(\tilde{\mathbf{x}}) = \operatorname{argmax}_{\tilde{\mathbf{x}} \in \mathbb{R}^p} \sum_{i=1}^n f_i(\tilde{\mathbf{x}}), \quad (1)$$

while being allowed to exchange information with neighbors only. These problems arise in decentralized control [1]–[4], sensor networks [5]–[7], and machine learning [8]–[10].

The theory and practice of first order methods to solve (1) is well developed. There are multiple methods that solve (1) in the primal domain [11]–[14] and a larger number of methods that solve (1) through duality theory [5], [7], [15]–[18]. However, and as is the case in centralized optimization, these first order methods are slow to converge when the objective function is ill-conditioned. This has motivated the development of decentralized second order methods which perform better than their first order counterparts, when the problems are not well conditioned and Hessians are available at reasonable computational cost [19]–[21]. Alas, evaluation

and inversion of Hessians is a task that can be computationally impractical in some problems. When this is the case in centralized optimization, the solution comes in the form of resorting to quasi-Newton methods [22]–[24]. The goal of this paper is to develop a decentralized quasi-Newton method to handle problems that are not well conditioned and in which second order information is not readily available.

We start by equating the solution of (1) to the minimization of a suitable dual function (Section II). A brief description of a regularized version of the centralized Broyden-Fletcher-Goldfarb-Shanno (BFGS) quasi-Newton method is then introduced (Section II-A). BFGS, regularized or not, can't be implemented in a decentralized manner because it relies on multiplying gradients by a curvature matrix that is not sparse. This limitation is overcome by the Decentralized (D-)BFGS method which relies on the observation that the appealing convergence traits of BFGS come from the curvature matrix satisfying a secant condition that can be expressed and satisfied in a decentralized manner (Section III). D-BFGS is a modification of regularized BFGS that maintains validity of this secant condition while ensuring the curvature matrix has a sparsity pattern matching the sparsity pattern of the graph. Asynchronous implementation of D-BFGS is further discussed (Section III-A). Convergence of D-BFGS is established for synchronous and asynchronous implementations (Section IV) and performance advantages relative to alternatives are evaluated numerically (Section V). We close the paper with concluding remarks (Section VI).

II. PROBLEM FORMULATION

We consider a decentralized system with n nodes which are connected as per the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = \{1, \dots, n\}$ is the set of nodes and $\mathcal{E} = \{(i, j)\}$ is the set of m edges. We assume the graph \mathcal{G} is symmetric, i.e., $(i, j) \in \mathcal{E}$ implies $(j, i) \in \mathcal{E}$, and the graph does not contain self-loops. Further, define the neighborhood of node i as the set $n_i := \{j \mid (i, j) \in \mathcal{E}\}$ of nodes j that are adjacent to i . The nodes have access to their local functions f_i only and their goal is to find the optimal argument $\mathbf{x}^* \in \mathbb{R}^p$ that minimizes the aggregate cost function, $\sum_{i=1}^n f_i(\tilde{\mathbf{x}})$ in (1). To rewrite this optimization problem in a manner that is more suitable for decentralized settings, we introduce the variable \mathbf{x}_i as a copy of the decision variable $\tilde{\mathbf{x}}$ kept at node i . We then rewrite the optimization problem in (1) as

$$\begin{aligned} \mathbf{x}^* &:= \operatorname{argmax}_{\mathbf{x}_1, \dots, \mathbf{x}_n} \sum_{i=1}^n f_i(\mathbf{x}_i) \\ \text{s.t.} \quad &\mathbf{x}_i = \mathbf{x}_j, \quad \text{for all } (i, j) \in \mathcal{E}. \end{aligned} \quad (2)$$

Work supported by NSF CAREER CCF-0952867 and ONR N00014-12-1-0997. Authors are with the Department of Electrical and Systems Engineering at the University of Pennsylvania, Philadelphia, PA 19104 USA. (email: maeisen, ariyanm, aribeiro@seas.upenn.edu).

Since the graph is connected, any feasible solution of (2) satisfies $\mathbf{x}_1 = \dots = \mathbf{x}_n$. With this restriction on the feasible set the cost functions in (1) and (2) become equivalent and the optimal argument $\mathbf{x}^* = \{\mathbf{x}_1^*, \dots, \mathbf{x}_n^*\}$ of (2) has the form $\mathbf{x}_1^* = \dots = \mathbf{x}_n^* = \tilde{\mathbf{x}}^*$.

We tackle the solution of (2) in the dual domain. Define then the dual variable $\lambda_{ij} \in \mathbb{R}^p$ associated with the constraint $\mathbf{x}_i = \mathbf{x}_j$ which is kept at node i . Moreover, define λ_i as the concatenation of all λ_{ij} for $j \in n_i$. Further, consider $\lambda := [\lambda_1; \dots; \lambda_n] \in \mathbb{R}^{mp}$ as the concatenation of the n dual variables λ_i , to write the Lagrangian $\mathcal{L}(\mathbf{x}, \lambda)$ of the optimization problem in (2) as

$$\mathcal{L}(\mathbf{x}, \lambda) = \sum_{i=1}^n f_i(\mathbf{x}_i) + \sum_{(i,j) \in \mathcal{E}} \lambda_{ij}^T (\mathbf{x}_i - \mathbf{x}_j). \quad (3)$$

Notice that the Lagrangian $\mathcal{L}(\mathbf{x}, \lambda)$ for a given dual vector λ is separable over the nodes. Hence, each node i computes the local Lagrangian maximizer $\mathbf{x}_i(\lambda)$ by solving the program

$$\mathbf{x}_i(\lambda) = \underset{\mathbf{x}_i \in \mathbb{R}^p}{\operatorname{argmax}} f_i(\mathbf{x}_i) + \sum_{j \in n_i} (\lambda_{ij} - \lambda_{ji})^T \mathbf{x}_i. \quad (4)$$

Upon defining the aggregate Lagrangian maximizer vector $\mathbf{x}(\lambda) := [\mathbf{x}_1(\lambda); \dots; \mathbf{x}_n(\lambda)]$ as the concatenation of the local maximizers $\mathbf{x}_i(\lambda)$, we can define the dual function as $h(\lambda) := \mathcal{L}(\mathbf{x}(\lambda), \lambda)$ and the dual problem as the minimization of the dual function,

$$\begin{aligned} \lambda^* &:= \underset{\lambda}{\operatorname{argmin}} h(\lambda) \\ &:= \underset{\lambda}{\operatorname{argmin}} \sum_{i=1}^n f_i(\mathbf{x}_i(\lambda)) + \sum_{(i,j) \in \mathcal{E}} \lambda_{ij}^T (\mathbf{x}_i(\lambda) - \mathbf{x}_j(\lambda)). \end{aligned} \quad (5)$$

For concave problems that satisfy minimal constraint qualifications that we assume to hold, the dual problem in (5) is equivalent to the primal problem in (2). In particular, the optimal primal variable at node i can be recovered as $\mathbf{x}_i(\lambda^*) = \mathbf{x}^*$ if the optimal multiplier λ^* is known [cf.(4)].

An important feature of the dual function $h(\lambda)$ is that its gradients can be computed locally as well. Specifically, it follows from the definition of the dual objective function $h(\lambda)$ in (5) that the partial derivative of $h(\lambda)$ with respect to the component λ_{ij} can be written as the constraint slack

$$\mathbf{g}_{ij}(\lambda) := \frac{\partial h(\lambda)}{\partial \lambda_{ij}} = \mathbf{x}_i(\lambda) - \mathbf{x}_j(\lambda). \quad (6)$$

Observe that to solve the program in (4), node i requires access to the local multipliers λ_{ij} and the dual variables λ_{ji} of its neighbors $j \in n_i$. Likewise, to evaluate the gradients $\mathbf{g}_{ij}(\lambda)$ in (6), node i needs access to the local Lagrangian maximizer $\mathbf{x}_i(\lambda)$ and the neighboring maximizer $\mathbf{x}_j(\lambda)$. It follows that gradient descent in the dual function can be implemented distributedly by relying on local operations and communication with neighboring nodes [7].

For future reference we emphasize that although the primal objective function f is strongly concave, the dual objective function h is not necessarily strongly convex. This fact requires the use of regularizations in the quasi-Newton

algorithm that we will propose in Section III. We study this regularization in the following section.

A. Regularized BFGS

Dual gradient descent can be implemented in a decentralized manner and proven to converge to optimal arguments. However, convergence is slow when the condition number of the dual function is large. In this paper we propose a *decentralized* quasi-Newton method to overcome this limitation. In *centralized* settings, the idea of quasi-Newton methods is to alter the descent direction by premultiplying the dual gradient with an approximation of its Hessian inverse. Specifically, consider a time index t and let $\lambda(t)$ denote the dual variable iterate at time t and $\mathbf{g}(t) = \mathbf{g}(\lambda(t))$ be the corresponding gradient. Further, introduce a step size $\epsilon(t)$ and a symmetric positive definite matrix $\mathbf{B}(t) \in \mathbb{R}^{mp \times mp}$ to define the dual quasi-Newton method through the recursion

$$\lambda(t+1) = \lambda(t) - \epsilon(t) \mathbf{B}(t)^{-1} \mathbf{g}(t) := \lambda(t) + \epsilon(t) \mathbf{d}(t), \quad (7)$$

where we have defined the descent direction $\mathbf{d}(t) := -\mathbf{B}(t)^{-1} \mathbf{g}(t)$ in the second equality. If we substitute the matrix $\mathbf{B}(t)$ by the dual Hessian $\nabla^2 h(\lambda(t))$, we recover the update of Newton's method. The idea of quasi-Newton methods is that to design the matrix $\mathbf{B}(t)$ as an approximation of the the dual Hessian $\nabla^2 h(\lambda(t))$, while avoiding the cost of its evaluation. Various quasi-Newton methods are known to accomplish this feat, with the most common being the method of Broyden-Fletcher-Goldfarb-Shanno (BFGS) [25]. To describe BFGS begin by defining the variable variation $\mathbf{v}(t)$ and the gradient variation $\mathbf{r}(t)$ as

$$\mathbf{v}(t) := \lambda(t+1) - \lambda(t), \quad \mathbf{r}(t) := \mathbf{g}(t+1) - \mathbf{g}(t). \quad (8)$$

The idea of *regularized* BFGS [26] is to find a matrix at each iteration $t+1$ that: (i) Satisfies the secant condition $\mathbf{B}(t+1) \mathbf{v}(t) = \mathbf{r}(t)$. (ii) Is closest to the previous Hessian approximation matrix $\mathbf{B}(t)$ with respect to a differential entropy measure. (iii) Has a smallest eigenvalue not smaller than a pre-specified constant γ . To express this matrix introduce the modified gradient variation vector $\tilde{\mathbf{r}}(t)$ with regularization constant $\gamma > 0$,

$$\tilde{\mathbf{r}}(t) := \mathbf{g}(t+1) - \mathbf{g}(t) - \gamma \mathbf{v}(t). \quad (9)$$

The Hessian approximation $\mathbf{B}(t)$ of regularized BFGS can then be computed by recursive application of

$$\mathbf{B}(t+1) = \mathbf{B}(t) + \frac{\tilde{\mathbf{r}}(t) \tilde{\mathbf{r}}(t)^T}{\tilde{\mathbf{r}}(t)^T \mathbf{v}(t)} - \frac{\mathbf{B}(t) \mathbf{v}(t) \mathbf{v}(t)^T \mathbf{B}(t)}{\mathbf{v}(t)^T \mathbf{B}(t) \mathbf{v}(t)} + \gamma \mathbf{I}. \quad (10)$$

The matrix $\mathbf{B}(t+1)$ in (10) is the closest to $\mathbf{B}(t)$ in terms of relative entropy among all the matrices that satisfy the original secant condition and whose smallest eigenvalue is at least γ [26, Proposition 1]. We emphasize that the use of the modified gradient variation in lieu of the regular gradient variation is necessary to maintain validity of the secant $\mathbf{B}(t+1) \mathbf{v}(t) = \mathbf{r}(t)$ condition which is the property that endows BFGS with appealing convergence traits; see [26] for details.

Both, the variable iteration in (7) and the matrix update in

(10), require centralized operations. In particular, to evaluate the inner product $\tilde{\mathbf{r}}(t)^T \mathbf{v}(t)$ in (10) or to compute the product $\mathbf{B}(t)^{-1} \mathbf{g}(t)$ in (7) nodes need access to global information. The goal of this paper is to introduce a variation of the regularized BFGS method that maintains the secant condition and is implementable in a decentralized manner.

III. DECENTRALIZED BFGS

We propose a decentralized implementation of BFGS (D-BFGS), in which each node approximates the curvature of its local cost function and its neighbors. In doing so, each node computes and stores a local Hessian inverse approximation. As in regularized BFGS, an important feature of D-BFGS is that while it can be formulated in a decentralized manner, the global descent of the algorithm still satisfies the original secant condition. To study the details, recall n_i as the neighborhood of node i and define $m_i := |n_i|$ as the number of neighbors of node i . We introduce the local dual vector $\boldsymbol{\lambda}_i(t) \in \mathbb{R}^{m_i p}$ of node i as the concatenation of the dual variables $\boldsymbol{\lambda}_{ij}(t)$ where $j \in n_i$. We use this definition to introduce the local variable variation at node i as

$$\mathbf{v}_i(t) := \frac{\boldsymbol{\lambda}_i(t+1) - \boldsymbol{\lambda}_i(t)}{m_i + 1}. \quad (11)$$

Nodes can exchange their local dual variable $\boldsymbol{\lambda}_i(t)$ with each other and construct a concatenated *neighborhood* dual variable $\boldsymbol{\lambda}_{n_i} \in \mathbb{R}^{M_i p}$ where $M_i = m_i + \sum_{j \in n_i} m_j$. The local variable variation in (11) can subsequently be extended to the neighborhood variable variation vector $\tilde{\mathbf{v}}_{n_i}(t) \in \mathbb{R}^{M_i p}$. Specifically, denote by $\mathbf{D}_{n_i} \in \mathbb{R}^{M_i p \times M_i p}$ the diagonal matrix such that its components corresponding to node j are $1/(m_j + 1)$. The modified neighborhood variable variation $\tilde{\mathbf{v}}_{n_i}(t)$ of node i is then given by

$$\tilde{\mathbf{v}}_{n_i}(t) = \mathbf{D}_{n_i} [\boldsymbol{\lambda}_{n_i}(t+1) - \boldsymbol{\lambda}_{n_i}(t)]. \quad (12)$$

Likewise, we can define the local gradient $\mathbf{g}_i(t) \in \mathbb{R}^{m_i p}$ at node i as the concatenation of the partial derivatives $\partial h(\boldsymbol{\lambda}) / \partial \boldsymbol{\lambda}_{ij} = \mathbf{x}_j^*(\boldsymbol{\lambda}) - \mathbf{x}_i^*(\boldsymbol{\lambda})$ for all $j \in n_i$. These can be then exchanged between neighboring nodes to construct a neighborhood gradient $\mathbf{g}_{n_i}(t) \in \mathbb{R}^{M_i p}$ and modified neighborhood gradient variation $\tilde{\mathbf{r}}_{n_i}(t) \in \mathbb{R}^{M_i p}$. With a regularization constant $\gamma > 0$, the modified neighborhood gradient variation $\tilde{\mathbf{r}}_{n_i}(t)$ is given by

$$\tilde{\mathbf{r}}_{n_i}(t) = \mathbf{g}_{n_i}(t+1) - \mathbf{g}_{n_i}(t) - \gamma \tilde{\mathbf{v}}_{n_i}(t). \quad (13)$$

Thus, we have defined the neighborhood variable variation $\tilde{\mathbf{v}}_{n_i}(t)$ and modified gradient variation $\tilde{\mathbf{r}}_{n_i}(t)$ so that they are suitable for decentralized settings. We introduce $\mathbf{B}^i(t)$ as the neighborhood Hessian approximation matrix, which is updated as

$$\begin{aligned} \mathbf{B}^i(t+1) &= \mathbf{B}^i(t) + \frac{\tilde{\mathbf{r}}_{n_i}(t) \tilde{\mathbf{r}}_{n_i}(t)^T}{\tilde{\mathbf{r}}_{n_i}(t)^T \tilde{\mathbf{v}}_{n_i}(t)} \\ &\quad - \frac{\mathbf{B}^i(t) \tilde{\mathbf{v}}_{n_i}(t) \tilde{\mathbf{v}}_{n_i}(t)^T \mathbf{B}^i(t)}{\tilde{\mathbf{v}}_{n_i}(t)^T \mathbf{B}^i(t) \tilde{\mathbf{v}}_{n_i}(t)} + \gamma \mathbf{I}. \end{aligned} \quad (14)$$

The update in (14) differs from (10) in its use of neighborhood variable and modified gradient variation as defined in

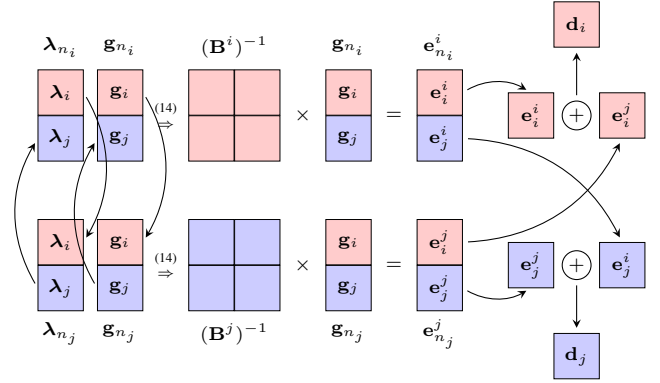


Fig. 1: D-BFGS variable flow. Nodes exchange variable and gradient variations – $\boldsymbol{\lambda}_i$ and \mathbf{g}_i sent to j and $\boldsymbol{\lambda}_j$ and \mathbf{g}_j sent to i – that they use to determine local curvature matrices – \mathbf{B}^i and \mathbf{B}^j . They then use exchanged gradients to compute descent directions – \mathbf{e}_i^i and \mathbf{e}_j^j . These contain a piece to add locally – \mathbf{e}_i^i stays at node i and \mathbf{e}_j^j stays at node j – and a piece to add at neighbors – \mathbf{e}_j^i is sent to node j and \mathbf{e}_i^j is sent to node i .

(12) and (13), respectively, instead of the variation vectors in (8) and (9).

We define the descent direction of D-BFGS with normalization constant $\Gamma > 0$ evaluated at node i as

$$\mathbf{e}_{n_i}^i(t) := -(\mathbf{B}^i(t)^{-1} + \Gamma \mathbf{D}_{n_i}) \mathbf{g}_{n_i}(t). \quad (15)$$

Note that $\Gamma \mathbf{D}_{n_i}$ is added to the Hessian inverse approximation $\mathbf{B}^i(t)^{-1}$ to ensure that the descent direction $\mathbf{e}_{n_i}^i(t)$ is not null.

The variable flow is demonstrated in Figure 1. Nodes exchange variable and gradient information to compute local Hessian inverse approximations $\mathbf{B}^i(t)^{-1}$ and neighborhood descent direction $\mathbf{e}_{n_i}^i(t)$. The descent direction $\mathbf{e}_{n_i}^i(t)$ contains a descent direction for node i and its neighbors $j \in n_i$. Nodes therefore exchange with their neighbors the parts of their locally computed descent direction pertaining to them. To be more precise, denote $\mathbf{e}_j^i(t) \in \mathbb{R}^{m_j p} = [\mathbf{e}_{n_i}^i(t)]_j$ as the component of the descent direction $\mathbf{e}_{n_i}^i(t)$ evaluated at node i that belongs to the neighbor j . Node i computes its full descent direction $\mathbf{d}_i(t)$ as the sum of locally computed descent directions $\mathbf{e}_i^i(t)$ and the parts received from its neighbors $\mathbf{e}_i^j(t)$, i.e.,

$$\mathbf{d}_i(t) := \mathbf{e}_i^i(t) + \sum_{j \in n_i} \mathbf{e}_i^j(t). \quad (16)$$

The local variable $\boldsymbol{\lambda}_i$ at node i is then updated using the full descent direction $\mathbf{d}_i(t)$ by

$$\boldsymbol{\lambda}_i(t+1) = \boldsymbol{\lambda}_i(t) + \epsilon(t) \mathbf{d}_i(t). \quad (17)$$

Note that at each step t the primal variable $\mathbf{x}_i(t+1)$ can subsequently be recovered as the Lagrangian maximizer with respect to $\boldsymbol{\lambda}(t+1)$, i.e.

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(\boldsymbol{\lambda}(t+1)). \quad (18)$$

The summary of the algorithm performed for node i is outlined in Algorithm 1. Each node begins with an initial

Algorithm 1 D-BFGS method at node i

Require: $\mathbf{B}^i(0), \boldsymbol{\lambda}_i(0), \mathbf{g}_i(0), \boldsymbol{\lambda}_{n_i}(0), \mathbf{g}_{n_i}(0)$

- 1: **for** $t = 0, 1, 2, \dots$ **do**
 - 2: Compute $\mathbf{e}_{n_i}^i(t) = -(\mathbf{B}^i(t)^{-1} + \Gamma \mathbf{D}_{n_i}) \mathbf{g}_{n_i}(t)$ [cf. (15)]
 - 3: Exchange $\mathbf{e}_{n_i}^i(t)$ with neighbors $j \in n_i$
 - 4: Compute descent dir. $\mathbf{d}_i(t) := \mathbf{e}_i^i(t) + \sum_{j \in n_i} \mathbf{e}_{n_i}^j(t)$.
 - 5: Update local variable $\boldsymbol{\lambda}_i(t+1) = \boldsymbol{\lambda}_i(t) + \epsilon(t) \mathbf{d}_i(t)$ [cf.(17)] and exchange with neighbors
 - 6: Compute $\mathbf{x}_i(\boldsymbol{\lambda}(t+1))$ and exchange with neighbors [cf. (4)]
 - 7: Compute $\mathbf{g}_{ij}(t+1)$ and exchange with neighbors [cf. (6)]
 $\mathbf{g}_{ij}(t+1) = \mathbf{x}_i(\boldsymbol{\lambda}(t+1)) - \mathbf{x}_j(\boldsymbol{\lambda}(t+1))$
 - 8: Compute $\tilde{\mathbf{v}}_{n_i}(t), \tilde{\mathbf{r}}_{n_i}(t), \mathbf{B}^i(t+1)$ [cf.(12)–(14)]
 - 9: **end for**
-

dual variable $\mathbf{v}_i(0)$, and Hessian approximation $\mathbf{B}^i(0)$, and gradient $\mathbf{g}_i(0)$. Nodes initially exchange local variable and gradients to construct initial neighborhood variable $\boldsymbol{\lambda}_{n_i}(0)$ and gradient $\mathbf{g}_{n_i}(0)$. For each step t , nodes compute their neighborhood descent direction $\mathbf{e}_{n_i}^i(t)$ in Step 3 and exchange the descent elements $\mathbf{e}_{n_i}^j(t)$ with their neighbors in Step 3 to compute the full descent direction $\mathbf{d}_i(t)$ as in Step 4. They use the full descent direction $\mathbf{d}_i(t)$ to update the variable $\boldsymbol{\lambda}_i(t+1)$ and exchange it with their neighbors to form $\boldsymbol{\lambda}_{n_i}(t+1)$ in Step 5. Then, they use their neighbor variables $\boldsymbol{\lambda}_j(t+1)$ to compute an updated Lagrangian maximizer $\mathbf{x}_i(\boldsymbol{\lambda}(t+1))$ in Step 6, which it then exchanges with its neighbors. With access to the Lagrangian maximizer $\mathbf{x}_i(\boldsymbol{\lambda}(t+1))$ of their neighbors, the gradient $\mathbf{g}_{ij}(t+1)$ can be updated and then exchanged as in Step 7. In Step 8, nodes compute their modified variable $\tilde{\mathbf{v}}_{n_i}(t)$ and gradient $\tilde{\mathbf{r}}_{n_i}(t)$ variations that are required for computing the updated neighborhood Hessian approximation matrix $\mathbf{B}^i(t+1)$.

Remark 1 We stress here the need in using the normalization matrix \mathbf{D}_{n_i} used in (12) for defining the variable variation $\mathbf{v}_{n_i}(t)$. Including \mathbf{D}_{n_i} in the definition ensures that the global Hessian inverse approximation matrix satisfies the global secant condition. To be more precise, consider the regularized neighborhood Hessian inverse approximation $\mathbf{B}^i(t)^{-1} + \Gamma \mathbf{I} \in \mathbb{R}^{M_i p \times M_i p}$. We define $\mathbf{H}^i(t) \in \mathbb{R}^{mp \times mp}$ to be the block sparse matrix with respect to n_i that has a dense sub-matrix $\mathbf{B}^i(t)^{-1}$. Further define $\hat{\mathbf{D}}_{n_i} \in \mathbb{R}^{mp \times mp}$ to be the block sparse matrix with respect to n_i that has a dense submatrix of \mathbf{D}_{n_i} . Considering this definition the descent direction $\mathbf{d}(t)$ for the global concatenated variable vector $\boldsymbol{\lambda}(t)$ can be written as

$$\begin{aligned} \mathbf{d}(t) &:= - \sum_{i=1}^n \left[\mathbf{H}^i(t) + \Gamma \hat{\mathbf{D}}_{n_i} \right] \mathbf{g}(t) \\ &:= - [\mathbf{H}(t) + \Gamma \mathbf{I}] \mathbf{g}(t). \end{aligned} \quad (19)$$

The expression in (19) states that the matrix $\mathbf{H}(t) := \sum_{i=1}^n \mathbf{H}^i(t)$ is the global Hessian inverse approximation. It is easily verifiable that the matrix $\mathbf{H}(t)$ satisfies the global

secant condition, i.e., $\mathbf{v}(t) = \mathbf{H}(t) \mathbf{r}(t)$ which justifies the normalization of the variable variation in (12).

Remark 2 As in the case of centralized BFGS, it is necessary that the neighborhood inner product $\tilde{\mathbf{r}}_{n_i}(t)^T \tilde{\mathbf{v}}_{n_i}(t)$ be positive in order for $\mathbf{B}^i(t)$ to be well defined. In the decentralized case, however, due to the truncating of the gradient and variable vectors we cannot guarantee that this inner product is positive even when the functions are strongly convex. As such, in practice the condition $\tilde{\mathbf{r}}_{n_i}(t)^T \tilde{\mathbf{v}}_{n_i}(t) > 0$ must be verified at every iteration, otherwise we do not update the Hessian approximation matrix, i.e. $\mathbf{B}^i(t+1) = \mathbf{B}^i(t)$. Note that for these iterations the local secant condition for node i is not satisfied.

A. Asynchronous implementation

Given the coordination and communication cost required to implement D-BFGS in Algorithm 1, we also consider the D-BFGS algorithm in the asynchronous setting. Our model for asynchronicity follows that used in [27], in which nodes perform computations and communications out of sync with their neighbors. Consider that the time indices are partitioned so that node i 's primary computation, namely the computation of descent direction $\mathbf{e}_{n_i}^i(t)$, requires multiple time iterates to complete. For each node i , we define a set $T^i \subseteq \mathbb{Z}^+$ of all time indices in which node i is available to send and receive information.

We further define two functions that specify the asynchronicity between nodes. For each node i , we define a function $\pi^i(t)$ that returns the most recent time node prior to t node i was available, i.e.

$$\pi^i(t) := \max\{\hat{t} \mid \hat{t} < t, \hat{t} \in T^i\}. \quad (20)$$

Moreover, we define a function $\pi_j^i(t)$ that returns the most recent time node j sent information that has been received by node i by time t , or explicitly,

$$\pi_j^i(t) := \pi^j(\pi^i(t)). \quad (21)$$

In the asynchronous setting, the superscript notation used to denote locally computed information now additionally signifies a node's current knowledge its neighbors, i.e.

$$\boldsymbol{\lambda}_j^i(t) := \boldsymbol{\lambda}_j(\pi_j^i(t)) \quad (22)$$

$$\boldsymbol{\lambda}_{n_i}^i(t) = [\boldsymbol{\lambda}_j^i(t)]_{j \in n_i}. \quad (23)$$

We emphasize that $\boldsymbol{\lambda}_j^i(t) \neq \boldsymbol{\lambda}_j^k(t)$ for any two nodes i and k at any time t . We consider as the current global variable state $\boldsymbol{\lambda}(t)$ the concatenation of each node's current knowledge of its own variable, i.e. $\boldsymbol{\lambda}(t) := [\boldsymbol{\lambda}_1^n(t); \dots; \boldsymbol{\lambda}_n^n(t)]$. We subsequently use the same notation for local gradients $\mathbf{g}_j^i(t)$ and descent directions $\mathbf{e}_j^i(t)$.

We assume at any time $t \in T^i$ that node i does three things: (i) It reads the variable, gradient, and descent directions from neighboring nodes $j \in n_i$ sent while it was busy. (ii) It updates its local variables and gradient using the descent direction it has just finished computing as well as the descent directions it has received from its neighbors.

Algorithm 2 Asynchronous D-BFGS method at node i

Require: $\mathbf{B}^i(0)$, $\lambda_i(0)$, $\mathbf{g}_i(0)$, $\mathbf{d}_{n_i}^i(0)$ [cf. (15)]

```
1: for  $t \in T^i$  do
2:   Read  $\mathbf{e}_i^j(t)$ ,  $\lambda_j^i(t)$ ,  $\mathbf{g}_i^j(t)$  from neighbors  $j \in n_i$ 
3:   Update  $\lambda_i(t+1)$ ,  $\mathbf{x}(\lambda(t))$ ,  $\mathbf{g}_i(t+1)$  [cf. (24), (25)]
4:   Compute  $\tilde{\mathbf{v}}_{n_i}^i(t)$ ,  $\tilde{\mathbf{r}}_{n_i}^i(t)$ ,  $\mathbf{B}^i(t+1)$  [cf. (26), (27), (14)]
5:   Compute  $\mathbf{e}_{n_i}^i(t+1)$  [cf. (15)]
6:   Send  $\lambda_i(t+1)$ ,  $\mathbf{g}_i(t+1)$ ,  $\mathbf{e}_i^j(t+1)$  to neighbors  $j \in n_i$ 
7: end for
```

(iii) Node i can send its locally computed descent direction as well as its updated variable and gradient info. To state in more explicit terms, node i performs the following update to its own variable at all times,

$$\lambda_i^i(t+1) = \lambda_i^i(t) + \epsilon(t)\mathbf{d}_i(t), \quad (24)$$

where $\mathbf{d}_i(t)$ is the decent for the i th variable $\mathbf{x}_i(t)$ at t ,

$$\mathbf{d}_i(t) = \begin{cases} \mathbf{e}_i^i(t) + \sum_{j \in n_i} \mathbf{e}_i^j(t) & \text{if } t \in T^i \\ \mathbf{0} & \text{otherwise.} \end{cases} \quad (25)$$

If $t \in T^i$, node i applies all descent directions available, otherwise it does nothing. Observe that the descent direction in (25) contains descents calculated with information from time $\pi^i(t)$ as well as the times $\pi_j^i(t)$ that neighbor j most recently updated its local variable.

To specify the asynchronous version of the D-BFGS algorithm, we reformulate the variable and gradient differences, $\tilde{\mathbf{v}}_{n_i}^i(t)$ and $\tilde{\mathbf{r}}_{n_i}^i(t)$ for the asynchronous case:

$$\tilde{\mathbf{v}}_{n_i}^i(t) = \mathbf{D}_{n_i} [\lambda_{n_i}^i(t+1) - \lambda_{n_i}^i(t)], \quad (26)$$

$$\tilde{\mathbf{r}}_{n_i}^i(t) = \mathbf{g}_{n_i}^i(t+1) - \mathbf{g}_{n_i}^i(t) - \gamma \tilde{\mathbf{v}}_{n_i}^i(t). \quad (27)$$

The computation of the local asynchronous update matrix $\mathbf{B}^i(t)$ and the corresponding descent direction $\mathbf{e}_{n_i}^i(t)$ follows respectively (14) and (15) exactly as in the synchronous setting, now using asynchronous $\tilde{\mathbf{v}}_{n_i}^i(t)$ and $\tilde{\mathbf{r}}_{n_i}^i(t)$ in place of $\tilde{\mathbf{v}}_{n_i}(t)$ and $\tilde{\mathbf{r}}_{n_i}(t)$, respectively.

The complete asynchronous algorithm is outlined in Algorithm 2. Each node begins with an initial variable $\lambda_i(0)$, Hessian approximation $\mathbf{B}^i(0)$, gradient $\mathbf{g}_i(0)$, and descent component $\mathbf{e}_i^i(0)$. At each time index t , they begin by reading the variables of neighbors $\mathbf{e}_i^j(t)$, $\lambda_j^i(t)$, $\mathbf{g}_i^j(t)$ in Step 2 and construct neighborhood variables. The aggregated descent direction $\mathbf{d}_i(t)$ is used to update variables $\lambda_i(t+1)$, $\mathbf{x}(\lambda(t+1))$, and $\mathbf{g}_i(t+1)$ in Step 3. Then, with the updated local variable $\lambda_i(t+1)$ and gradient $\mathbf{g}_i(t+1)$, the node computes the D-BFGS variables $\tilde{\mathbf{v}}_{n_i}^i(t)$, $\tilde{\mathbf{r}}_{n_i}^i(t)$, and $\mathbf{B}^i(t+1)$ in Step 4. In Step 5, node i computes the next descent direction $\mathbf{d}_{n_i}^i(t+1)$, and sends its variables to neighbors in Step 6.

While Algorithm 2 follows a similar structure to the synchronous Algorithm 1, we highlight that in the synchronous algorithm, coordination of four rounds of communication were required at each iteration of Algorithm 1 to properly communicate the dual variable, primal variable, and dual gradient information. In the asynchronous setting, only a

single round of communication is possible at each time iteration, thus the communication burden is indeed reduced and the coordination between nodes rendered unnecessary.

IV. CONVERGENCE ANALYSIS

In this section, we study the convergence properties of the D-BFGS method and show that the sequence of primal iterates \mathbf{x}_i generated by D-BFGS converges to the optimal argument $\tilde{\mathbf{x}}^*$ of (1). In proving these results we make the following assumptions.

Assumption 1 *The local objective functions $f_i(\mathbf{x})$ are differentiable and strongly convex with parameter $\mu > 0$.*

The strong convexity of local functions f_i implies that the aggregate function $f(\mathbf{x}) = \sum_{i=1}^n f_i(\mathbf{x}_i)$ is also strongly convex with constant μ . Define the oriented incidence matrix $\mathbf{A} \in \mathbb{R}^{mp \times n}$ where the A_{ev} component is 1 if edge e is started from node v and is -1 if edge e is ended at node v ; otherwise $A_{ev} = 0$. It is not difficult to see that the Hessian of the dual function can be written as $\nabla^2 h(\lambda) = \mathbf{A}(\nabla^2 f(\mathbf{x}(\lambda)))^{-1} \mathbf{A}^T$ from where we conclude that the eigenvalues of the dual function Hessian are upper bounded by $4n/\mu$ (note: $\mathbf{A}^T \mathbf{A}$ is two times the graph Laplacian matrix). In turn, this implies that the dual function gradients $\mathbf{g}(\lambda)$ are Lipschitz continuous with constant $4n/\mu$,

$$\|\mathbf{g}(\lambda) - \mathbf{g}(\tilde{\lambda})\| \leq \frac{4n}{\mu} \|\lambda - \tilde{\lambda}\|. \quad (28)$$

We additionally make a further assumption regarding the inner product of neighborhood variable and gradient variations.

Assumption 2 *For all i and t , the inner product between the neighborhood modified variable and gradient vector variations is strictly positive, i.e. $\tilde{\mathbf{v}}_{n_i}^T \tilde{\mathbf{r}}_{n_i} > 0$.*

This assumption is necessary to ensure all local Hessian approximations are well defined in (14). While this assumption does not always hold in practice, we use it regardless to simplify analysis. We stress that, in the case the assumption is violated, setting $\mathbf{B}^i(t+1) = \mathbf{B}^i(t)$ (See Remark 2) does not have any bearing on the proceeding analysis.

We specify $\mathbf{H}(t) = \sum_{i=1}^n \mathbf{H}^i(t)$ as the global Hessian inverse approximation and $\mathbf{d}(t) = -[\mathbf{H}(t) + \Gamma \mathbf{I}]\mathbf{g}(t)$ as the global descent direction. The following lemma establishes the positive definiteness of the global descent direction.

Lemma 1 *Consider the D-BFGS method introduced in (11)-(19). Further, recall both the positive constants γ and Γ as the regularization parameters of D-BFGS and the definition of the global Hessian inverse approximation $\mathbf{H}(t) + \Gamma \mathbf{I} = \sum_{i=1}^n [\mathbf{H}^i(t) + \Gamma \mathbf{D}_{n_i}]$. The eigenvalues of the global Hessian inverse approximation $\mathbf{H}(t)$ are uniformly bounded as*

$$\Gamma \mathbf{I} \preceq \mathbf{H}(t) + \Gamma \mathbf{I} \preceq \Delta \mathbf{I} := \left(\Gamma + \frac{n}{\gamma} \right) \mathbf{I}, \quad (29)$$

where n is the size of network.

Proof: The lower bound on $\mathbf{H}(t) + \Gamma\mathbf{I}$ follows immediately from the fact that $\mathbf{H}(t)$ is a sum of positive semidefinite matrices and is therefore a positive semidefinite matrix with eigenvalues greater than or equal to 0. The upper bound subsequently follows from the fact that each $\mathbf{H}_i(t)$ have eigenvalues upper bounded by $1/\gamma$, as the dense submatrix $\mathbf{B}^i(t)^{-1} \preceq 1/\gamma\mathbf{I}$. Then, the sum of n such matrices recovers the upper bound in (29). ■

The result in Lemma 1 shows that the eigenvalues of the global Hessian inverse approximation $\mathbf{H}(t)$ are uniformly bounded. Thus, D-BFGS descent direction $\mathbf{d}(t) = -\mathbf{H}(t)\mathbf{g}(t)$ is a valid descent direction. We use this result and convexity of the dual function h to show that the sequence of dual objective function errors $h(\boldsymbol{\lambda}) - h(\boldsymbol{\lambda}^*)$ converges to null.

Theorem 1 *Consider the D-BFGS method introduced in (11)-(19). If Assumption 1 holds true and $\epsilon(t)$ is chosen such that $\epsilon(t) < \Gamma\mu/(n\Delta^2)$, then the dual objective function error $h(\boldsymbol{\lambda}(t)) - h(\boldsymbol{\lambda}^*)$ converges to zero at least in the order of $o(1/t)$, i.e.,*

$$h(\boldsymbol{\lambda}(t)) - h(\boldsymbol{\lambda}^*) \leq o\left(\frac{1}{t}\right). \quad (30)$$

The proof for this result is standard, and is a small variation of that for gradient descent found in [28, Proposition 1.3.3]. Theorem 1 shows that the sequence of the dual objective function $h(\boldsymbol{\lambda})$ generated by D-BFGS converges to the optimal dual function value $h(\boldsymbol{\lambda}^*)$. We conclude with a corollary establishing the convergence of the primal function error and the primal variables of the original problem in (2).

Corollary 1 *The sequence of primal function value error generated by the D-BFGS algorithm converges to zero, i.e.*

$$\lim_{t \rightarrow \infty} f(\mathbf{x}^*) - f(\mathbf{x}(t)) = 0. \quad (31)$$

Furthermore, the sequence of primal variables $\mathbf{x}(t)$ converges to the optimal primal variable \mathbf{x}^* at least in the order of $o(1/\sqrt{t})$, i.e.

$$\|\mathbf{x}(t) - \mathbf{x}^*\| \leq o\left(\frac{1}{\sqrt{t}}\right) \quad (32)$$

Proof: See Appendix I. ■

We subsequently further provide a convergence result for the asynchronous implementation of DBFGS. The result uses the common assumption of partial asynchronicity, i.e. any two nodes are no more than some constant B out of sync. We demonstrate convergence in the following theorem.

Theorem 2 *Consider the asynchronous D-BFGS algorithm proposed in (24)-(27) and (14)-(16). If Assumptions 1 holds and the following partial asynchronicity property is satisfied for some $B > 0$,*

$$\max\{0, t - B + 1\} \leq \pi_j^i(t) \leq t, \quad \text{for all } i, j, t, \quad (33)$$

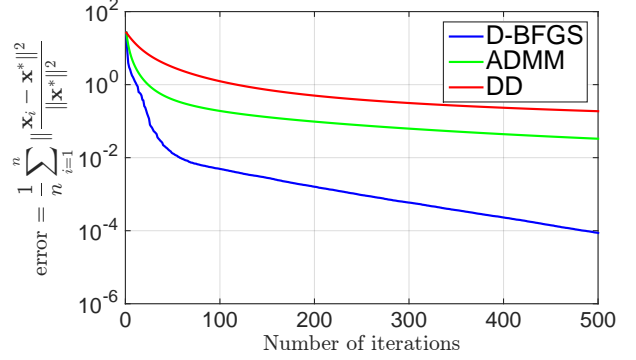


Fig. 2: Convergence path of the average distance to optimal primal variable vs. number of iterations for D-BFGS, ADMM, and DD for quadratic problem with condition number 10^2 .

then there exists a stepsize $\epsilon(t)$ such that $\lim_{t \rightarrow \infty} \mathbf{g}(t) = 0$.

The proof for this result is omitted for space considerations and can be found in [29, Theorem 3]. This theorem demonstrates that the uncoordinated and asynchronous implementation of DBFGS is still guaranteed to converge.

V. NUMERICAL RESULTS

We provide numerical results of the performance of D-BFGS and first order methods ADMM [5] and dual descent (DD) in solving a quadratic program. Consider the problem

$$\mathbf{x}^* := \operatorname{argmax}_{\mathbf{x} \in \mathbb{R}^p} \sum_{i=1}^n -\frac{1}{2} \mathbf{x}^T \mathbf{A}_i \mathbf{x} - \mathbf{b}_i^T \mathbf{x}, \quad (34)$$

where $\mathbf{A}_i \in \mathbb{R}^{p \times p}$ is the positive definite matrix and $\mathbf{b}_i \in \mathbb{R}^p$ is a random vector which are both available only at node i . In this case the Lagrangian maximizer and primal update in (4) can be computed with a closed form solution as

$$\mathbf{x}_i(\boldsymbol{\lambda}) = \mathbf{A}_i^{-1} \sum_{j \in n_i} (\lambda_{ji} - \lambda_{ij}) + \mathbf{A}_i^{-1} \mathbf{b}_i. \quad (35)$$

The rest of the D-BFGS updates for this problem follow from (6) and (11)-(19).

To both ensure the local objective functions are concave and control the problem's condition number, we set the matrices $\mathbf{A}_i := \operatorname{diag}\{\mathbf{A}_i\}$. The first $p/2$ elements \mathbf{a}_i are randomly chosen from the interval $[1, 10^{-1}]$ and the last $p/2$ elements are chosen randomly from the interval $[1, 10^1]$. The resulting $\mathbf{A} = \sum_{i=1}^n \mathbf{A}_i$ is then a positive definite with a condition number of 10^2 . For the vectors \mathbf{b}_i , the elements are chosen uniformly and randomly from the box $[0, 1]^p$. In our simulations we fix the variable dimension $p = 4$ and the number of nodes $n = 50$. The regularization parameters for D-BFGS are chosen to be $\gamma = 10^{-2}$ and $\Gamma = 10^{-3}$. In the experiments, the step size for each method is chosen to be constant and attempt is made to choose the largest step size for which the algorithms are observed to converge. For the network structure, we consider a 4-regular cycle graph.

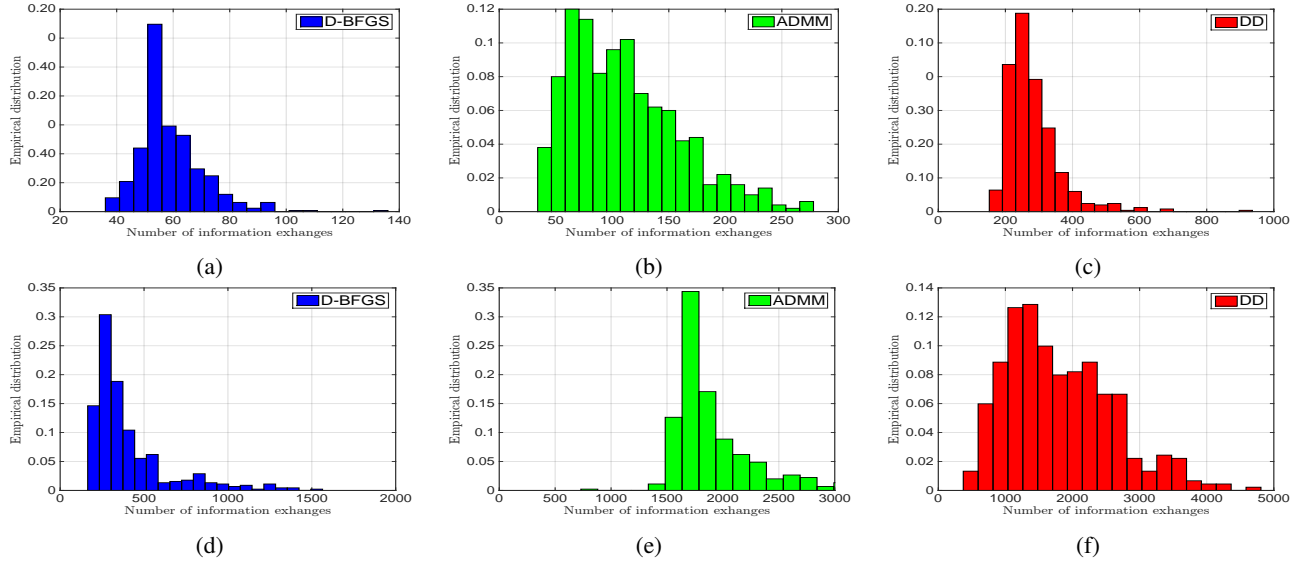


Fig. 3: Histogram of number of local communications needed to converge for D-BFGS, ADMM, and DD for condition numbers of (a)-(c) 10^0 and (d)-(f) 10^2 . In all cases, D-BFGS provides significant improvement in convergence time over first order methods, with the larger improvement for larger condition number.

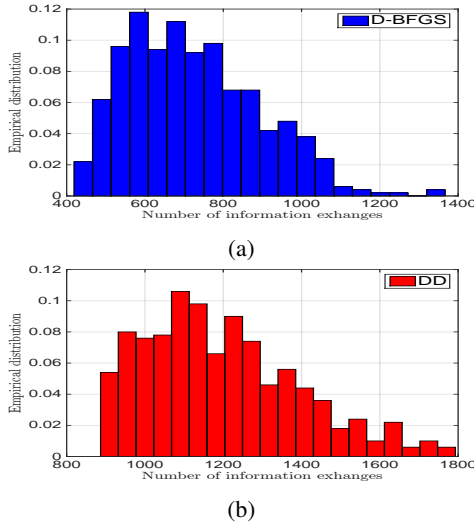


Fig. 4: Histogram of number of local communications needed to converge for asynchronous (a) D-BFGS and (b) DD. D-BFGS provides significant improvement in convergence time over DD.

We simulate the performance of D-BFGS, ADMM, and DD on the dual problem in (5) each using respective step sizes of 0.01, 0.002, and 0.002. To view convergence with respect to the original primal problem, we look directly at how fast and how close the algorithms reach the optimal point in the original primal formulation rather than looking at the norm of dual gradient. The optimal point \mathbf{x}^* is calculated for the quadratic problem in (34) using the closed form solution of a quadratic problem with linear constraints [30] and we evaluate the normalized average error as

$$e(t) := \frac{1}{n} \sum_{i=1}^n \frac{\|\mathbf{x}_i(t) - \mathbf{x}^*\|^2}{\|\mathbf{x}^*\|^2}. \quad (36)$$

Figure 2 shows the normalized average error $e(t)$ for all three algorithms with respect to the number of iterations. We see that D-BFGS converges substantially faster than the first order methods, reaching an average distance of 8.7×10^{-5} by iteration 500, while ADMM and DD just reach 3.3×10^{-2} and 1.8×10^{-1} respectively by iteration 500.

It is worth noting that D-BFGS requires four local exchanges with neighbors per iteration, while ADMM and DD require only two. In Figure 3 we thus present a histogram of the number of local exchanges required for each algorithm to converge, which we define as $\delta(t) = 10^{-2}$, over the course of 1000 independent trials for both small and large condition numbers. We see that D-BFGS requires about a factor of 2 and 5 less than ADMM and DD respectively for a small condition number of 10^0 . With larger condition number, however, the improvement of D-BFGS increases to close to a factor of 7 and 8 respectively. These results showcase the advantages of decentralized quasi-Newton methods over first-order gradient descent methods.

We additionally perform a numerical analysis on both D-BFGS and DD in the asynchronous setting for condition number 10^0 . To generate asynchronicity, we employ a simple model of aggregated Gaussian drift between nodes' local clocks. In Figure 4, we present histograms of number of information exchanges required to converge to an average of error of 5×10^{-2} for both D-BFGS and DD with respective stepsizes of 0.007 and 0.001. While a degradation in convergence time relative to synchronous algorithms is clearly evident, D-BFGS nonetheless continues to outperform DD, requiring on average about 600 and 1200 exchanges to occur respectively.

VI. CONCLUSIONS

We considered the problem of decentralized consensus optimization, in which nodes sought to maximize an aggregate

cost function while only being aware of a local strictly concave component. The problem was solved in the dual domain through the introduction of D-BFGS as a decentralized quasi-Newton method. In D-BFGS, a node approximates the curvature of its local cost function and its neighboring nodes to correct its descent direction. Analytical and numerical results were established showing its convergence and improvement over decentralized gradient descent methods, respectively, in synchronous and asynchronous settings.

APPENDIX I PROOF OF COROLLARY 1

The first result follows from the strong concavity of the primal function f , which implies that the duality gap is zero. The second result follows from the argument in [31, Theorem 1], repeated here.

Consider the Lagrangian function $\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda})$ in (3), which is strongly concave with respect to \mathbf{x} with parameter μ for strongly concave f_i 's with parameter μ . It is then the case that the following inequality holds for two points, the Lagrangian maximizer $\mathbf{x}(\boldsymbol{\lambda})$ [cf. (4)] and the optimal primal variable \mathbf{x}^* ,

$$\mathcal{L}(\mathbf{x}(\boldsymbol{\lambda}), \boldsymbol{\lambda}) - \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}) \geq \frac{\mu}{2} \|\mathbf{x}(\boldsymbol{\lambda}) - \mathbf{x}^*\|^2. \quad (37)$$

Observe that by using the matrix \mathbf{A} defined in Section IV, we can rewrite the Lagrangian function as $\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{A}\mathbf{x}$. We can find an upper bound on $\mathcal{L}(\mathbf{x}(\boldsymbol{\lambda}), \boldsymbol{\lambda}) - \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda})$ with respect to the dual function as

$$\begin{aligned} \mathcal{L}(\mathbf{x}(\boldsymbol{\lambda}), \boldsymbol{\lambda}) - \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}) &= f(\mathbf{x}(\boldsymbol{\lambda})) + \boldsymbol{\lambda}^T \mathbf{A}\mathbf{x}(\boldsymbol{\lambda}) \\ &\quad - f(\mathbf{x}^*) - \boldsymbol{\lambda}^T \mathbf{A}\mathbf{x}^*. \end{aligned} \quad (38)$$

The first two terms in (38) are equivalent to the dual function $h(\boldsymbol{\lambda})$, while $f(\mathbf{x}^*) = h(\boldsymbol{\lambda}^*)$ because the duality gap is zero. Additionally $\mathbf{A}\mathbf{x}^* = \mathbf{0}$ by construction so (38) reduces to

$$\mathcal{L}(\mathbf{x}(\boldsymbol{\lambda}), \boldsymbol{\lambda}) - \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}) = h(\boldsymbol{\lambda}) - h(\boldsymbol{\lambda}^*). \quad (39)$$

We can then combine the results of (37), (39), and (30) to obtain

$$\frac{\mu}{2} \|\mathbf{x}(\boldsymbol{\lambda}) - \mathbf{x}^*\|^2 \leq h(\boldsymbol{\lambda}) - h(\boldsymbol{\lambda}^*) \leq o\left(\frac{1}{t}\right), \quad (40)$$

which provides us the with the result in (32).

REFERENCES

- [1] R. Olfati-Saber and R. M. Murray, "Consensus problems in networks of agents with switching topology and time-delays," *Automatic Control, IEEE Transactions on*, vol. 49, no. 9, pp. 1520–1533, 2004.
- [2] F. Bullo, J. Cortés, and S. Martinez, *Distributed control of robotic networks: a mathematical approach to motion coordination algorithms*. Princeton University Press, 2009.
- [3] Y. Cao, W. Yu, W. Ren, and G. Chen, "An overview of recent progress in the study of distributed multi-agent coordination," *IEEE Transactions on Industrial Informatics*, vol. 9, pp. 427–438, 2013.
- [4] C. G. Lopes and A. H. Sayed, "Diffusion least-mean squares over adaptive networks: Formulation and performance analysis," *Signal Processing, IEEE Transactions on*, vol. 56, no. 7, pp. 3122–3136, 2008.
- [5] I. D. Schizas, A. Ribeiro, and G. B. Giannakis, "Consensus in ad hoc wsn with noisy links—part i: Distributed estimation of deterministic signals," *Signal Processing, IEEE Transactions on*, vol. 56, no. 1, pp. 350–364, 2008.
- [6] U. A. Khan, S. Kar, and J. M. Moura, "Diland: An algorithm for distributed sensor localization with noisy distance measurements," *Signal Processing, IEEE Transactions on*, vol. 58, no. 3, pp. 1940–1947, 2010.
- [7] M. Rabbat and R. Nowak, "Distributed optimization in sensor networks," in *Proceedings of the 3rd international symposium on Information processing in sensor networks*. ACM, 2004, pp. 20–27.
- [8] R. Bekkerman, M. Bilenko, and J. Langford, *Scaling up machine learning: Parallel and distributed approaches*. Cambridge University Press, 2011.
- [9] K. I. Tsianos, S. Lawlor, and M. G. Rabbat, "Consensus-based distributed optimization: Practical issues and applications in large-scale machine learning," *Communication, Control, and Computing (Allerton), 2012 50th Annual Allerton Conference on*, pp. 1543–1550, 2012.
- [10] V. Cevher, S. Becker, and M. Schmidt, "Convex optimization for big data: Scalable, randomized, and parallel algorithms for big data analytics," *Signal Processing Magazine, IEEE*, vol. 31, no. 5, pp. 32–43, 2014.
- [11] A. Nedic and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *Automatic Control, IEEE Transactions on*, vol. 54, no. 1, pp. 48–61, 2009.
- [12] A. Nedić, A. Ozdaglar, and P. A. Parrilo, "Constrained consensus and optimization in multi-agent networks," *Automatic Control, IEEE Transactions on*, vol. 55, no. 4, pp. 922–938, 2010.
- [13] K. Yuan, Q. Ling, and W. Yin, "On the convergence of decentralized gradient descent," *arXiv preprint arXiv:1310.7063*, 2013.
- [14] W. Shi, Q. Ling, G. Wu, and W. Yin, "Extra: An exact first-order algorithm for decentralized consensus optimization," *SIAM Journal on Optimization*, vol. 25, no. 2, pp. 944–966, 2015.
- [15] N. Chatzianagiotis, D. Datcheva, and M. M. Zavlanos, "An augmented lagrangian method for distributed optimization," *Mathematical Programming*, pp. 1–30, 2013.
- [16] R. T. Rockafellar, "Augmented lagrangians and applications of the proximal point algorithm in convex programming," *Mathematics of operations research*, vol. 1, no. 2, pp. 97–116, 1976.
- [17] D. Jakovetic, J. M. Moura, and J. Xavier, "Linear convergence rate of a class of distributed augmented lagrangian algorithms," *Automatic Control, IEEE Transactions on*, vol. 60, no. 4, pp. 922–936, 2015.
- [18] D. Jakovetic, J. Xavier, and J. M. Moura, "Fast distributed gradient methods," *Automatic Control, IEEE Transactions on*, vol. 59, no. 5, pp. 1131–1146, 2014.
- [19] A. Mokhtari, Q. Ling, and A. Ribeiro, "Network newton-part i: Algorithm and convergence," *arXiv preprint arXiv:1504.06017*, 2015.
- [20] D. Bajovic, D. Jakovetic, N. Krejic, and N. K. Jerinkic, "Newton-like method with diagonal correction for distributed optimization," *arXiv preprint arXiv:1509.01703*, 2015.
- [21] A. Mokhtari, W. Shi, Q. Ling, and A. Ribeiro, "A decentralized second-order method with exact linear convergence rate for consensus optimization," *arXiv preprint arXiv:1602.00596*, 2016.
- [22] C. G. Broyden, J. E. D. Jr., Wang, and J. J. More, "On the local and superlinear convergence of quasi-newton methods," *IMA J. Appl. Math.*, vol. 12, no. 3, pp. 223–245, June 1973.
- [23] R. H. Byrd, J. Nocedal, and Y. Yuan, "Global convergence of a class of quasi-newton methods on convex problems," *SIAM J. Numer. Anal.*, vol. 24, no. 5, pp. 1171–1190, October 1987.
- [24] L. Dong C. and J. Nocedal, "On the limited memory bfgs method for large scale optimization," *Mathematical programming*, no. 45(1-3), pp. 503–528, 1989.
- [25] J. Nocedal and S. Wright, *Numerical optimization*. Springer Science & Business Media, 2006.
- [26] A. Mokhtari and A. Ribeiro, "Res: Regularized stochastic bfgs algorithm," *Signal Processing, IEEE Transactions on*, vol. 62, no. 23, pp. 6089–6104, 2014.
- [27] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and distributed computation: numerical methods*. Prentice-Hall, Inc., 1989.
- [28] D. P. Bertsekas, "Nonlinear programming," 1999.
- [29] M. Eisen, A. Mokhtari, and A. Ribeiro, "Decentralized quasi-newton methods," 2016, available at <http://www.seas.upenn.edu/~maeisen/wiki/dbfgs.pdf>.
- [30] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.
- [31] A. Beck, A. Nedic, A. Ozdaglar, and M. Teboulle, "An gradient method for network resource allocation problems," *Control of Network Systems, IEEE Transactions on*, vol. 1, no. 1, pp. 64–73, 2014.